APPENDIX

ImageIn Device Manager

The ImageIn device manager provides interfaces for enumerating devices, querying properties of installed ImageIn devices and creating ImageIn device objects. The device manager has three objects.

CImageInDevMgr is used to:
- Create an enumerator object, CEnumImageInDevInfo
- Create a device object when given a DeviceID.
- Display UI to let a user choose a device object.
- Display UI to both choose a device and acquire an image from the chosen device.

CEnumImageInDevInfo is used to:
- Enumerate all ImageIn devices on a system. For each device enumerated, a CImageInDevInfo object is returned.

CImageInDevInfo is used to:
- Query device information properties from the ImageIn device. One of the properties, Device ID, can be used by CImageInDevMgr to create a device object.

Device information properties are distinct from normal device properties in that they are read from the registry and can be queried without forcing the device object to be created.
A program using its own methods to select an ImageIn device would do the following:
- Create a CImageInDevMgr object
- Use the IImageInDevMgr interface to create a CEnumImageInDevInfo object
- Use IEnumIMAGEIN_DEV_INFO interface to create a CImageInDevInfo object for each ImageIn device enumerated.
- Use IPropertyStorage interface of CImageInDevInfo object to inspect the device information properties of each device, save the Device ID property from the desired device.
- Use the DeviceID property in IImageInDevMgr interface to create an ImageIn device object.

IImageInDevMgr Interface

EnumImageInDevInfo

```
HRESULT IImageInDevMgr::EnumImageInDevInfo(
      LONG                          lFlags,
      IEnumIMAGEIN_DEV_INFO**       ppIEnum);
```

EnumImageInDevInfo creates a standard enumerator for CImageInDevInfo objects. The enumerator itself is a CEnumImageInDevInfo object that has a single interface IEnumIMAGEIN_DEV_INFO. Applications can use this API to obtain device information properties from available ImageIn devices.

Parameters:

lFlags	Specifies the type of device to enumerate.
ppIEnum	Enumeration interface for CEnumImageInDevInfo objects.

GetImageInDevInfo

```
HRESULT IImageInDevMgr::GetImageInDevInfo(
      BSTR                          bstrDeviceID,
      IPropertyStorage**            ppIPropStg);
```

Given a device ID GetImageInDevInfo returns an IPropertyStorage interface to CImageInDevInfo objects. This API gives applications a quick way to retrieve device information properties from a stored device ID.

Parameters:

bstrDeviceID,	Device ID string returned from device info object.
ppIPropStg	IPropertyStorage interface to CImageInDevInfo objects.

CreateImageInDev

```
HRESULT IImageInDevMgr::CreateImageInDev(
      BSTR                    bstrDeviceID,
      IUnknown**              ppDevice);
```

CreateImageInDev creates a device object identified by bstrDeviceID and returns an IUnknown interface to the object. The application can use QueryInterface on this IUnknown to access the other interfaces on a device.

Parameters:

bstrDeviceID	Device ID string returned from device info object.
ppDevice	IUnknown interface pointer returned.

SelectImageInDev

```
HRESULT IImageInDevMgr::SelectImageInDev(
      LONG                    lFlags,
      IUnknown**              ppDevice);
```

SelectImageInDev displays UI for the selection of ImageIn devices. When the user selects a device, the device object is created and an IUnknown interface is returned. The application can then use QueryInterface on this IUnknown to access the other interfaces on a device. Returns S_FALSE if the user cancels the selection dialog without making a device selection, S_OK if the user makes a device selection or an error value if the method fails.

Parameters:

lFlags	Operation flags
ppDevice	IUnknown interface pointer returned

GetImage

```
HRESULT IImageInDevMgr::GetImage(
      LONG                    lFlags,
      LONG                    lIntent,
      LONG                    cfFormat,
      STGMEDIUM*              pMedium);
```

GetImage displays device selection UI allowing a user to select a device. If the user selects a device, a device is created and device UI is displayed to capture the image. Image data is returned in the pMedium structure. Image format and default image capture properties may be specified using the lIntent and cfFormat parameters. Returns S_FALSE if the user cancels the selection dialog without making a device selection, S_OK if the user makes a device selection and the data is transferred or an error value if the method fails.

Parameters:

lFlags	Operation flags
lIntent	Intent, default image capture properties
cfFormat	Clipboard format desired by application
pMedium	Image is returned through this medium.

EnumDestinationInfo

```
HRESULT IImageInDevMgr::EnumDestinationInfo(
      LONG                          lFlags,
      IUnknown*                     pDevice,
      IEnumIMAGEIN_DEST_INFO**      ppIEnum);
```

EnumImageInDestInfo creates a standard enumerator for CImageInDestInfo objects. The enumerator itself is a CEnumImageInDestInfo object that has a single interface IEnumIMAGEIN_DEST_INFO. Applications can use this API to obtain device destination properties from an ImageIn device. Destination applications are registered using RegisterDestinationApplication.

Parameters:

lFlags	Enumeration flags.
pDevice	Device object. If null destination applications for all devices are enumerated.
ppIEnum	Enumeration interface for CEnumImageInDestInfo objects.

RegisterDestinationApplication

```
HRESULT IImageInDevMgr::RegisterDestinationApplication(
      LONG                    lFlags,
      LONG                    lIntent,
      IUnknown*               pDevice,
      BSTR                    bstrEvent,
      BSTR                    bstrAppName,
      BSTR                    bstrCommandLine);
```

This method is called when an application wishes to register as an ImageIn destination.

Parameters:

lFlags	Registration flags.
lIntent	Intent, default image capture properties.
pDevice	Device object. If null, destination application is registered for all devices.
bstrEvent	Optional event name.
bstrAppName	Friendly name of Application. This name will be displayed to the user in UI.
bstrCommandLine	Full path to the executable for this application. Additional command line arguments may be added to this command.

UnregisterDestinationApplication

```
HRESTUL IImageInDevMgr::UnregisterDestinationApplication
      LONG                          lFlags,
      IUnknown*                     pDevice,
      BSTR                          bstrAppName);
```

This method is called when an application that has registered an ImageIn destination wishes to be uninstalled or no longer known as an ImageIn destination

Parameters:

lFlags	Operation flags.
pDevice	Device object. If null, destination application is unregistered for all devices.
bstrAppName	Friendly name of Application, used to register application.

IEnumIMAGEIN_DEV_INFO Interface

The IEnumIMAGEIN_DEV_INFO interface is a standard OLE enumeration interface that supports per device enumeration of device information properties.

Next

```
HRESULT IEnumIMAGEIN_DEV_INFO::Next(
        ULONG                 celt,
        IPropertyStorage**    rgelt,
        ULONG                 *pceltFetched);
```

Next returns an array of IPropertyStorage interfaces to CImageInDevInfo objects. Applications can use the returned of IPropertyStorage interfaces to obtain device information properties from available ImageIn devices.

Parameters:

celt	Specifies the number of elements requested.
rgelt	Address of an array in which to return the IPropertyStorage interfaces.
pceltFetched	Address of a value that receives a count of the item identifiers actually returned in *rgelt*.

Skip

```
HRESULT IEnumIMAGEIN_DEV_INFO::Skip(ULONG celt);
```

Skip skips device objects in the enumeration.

Parameters:

celt	Number of items to skip.

Reset

```
HRESULT IEnumIMAGEIN_DEV_INFO::Reset(void);
```

Reset sets the enumeration back to the first device.

Clone

```
HRESULT IEnumIMAGEIN_DEV_INFO::Clone(
      IEnumIMAGEIN_DEV_INFO **ppIEnum);
```

Clone creates another IEnumIMAGEIN_DEV_INFO enumeration object and returns an interface pointer to it.

Parameters:

ppIEnum	Address that receives a pointer to the new enumeration object.

IEnumIMAGEIN_DEST_INFO Interface

The IEnumIMAGEIN_DEST_INFO interface is a standard OLE enumeration interface that supports enumeration of device destination properties.

Next

```
HRESULT IEnumIMAGEIN_DEST_INFO::Next(
        ULONG                celt,
        IPropertyStorage**   rgelt,
        ULONG                *pceltFetched);
```

Next returns an array of IPropertyStorage interfaces to CImageInDestInfo objects. Applications can use the returned of IPropertyStorage interfaces to obtain destination information properties from available ImageIn devices.

Parameters:

celt	Specifies the number of elements requested.
rgelt	Address of an array in which to return the IPropertyStorage interfaces.
pceltFetched	Address of a value that receives a count of the item identifiers actually returned in *rgelt*.

Skip

```
HRESULT IEnumIMAGEIN_DEST_INFO::Skip(ULONG celt);
```

Skip skips destination objects in the enumeration.

Parameters:

celt	Number of items to skip.

Reset

```
HRESULT IEnumIMAGEIN_DEST_INFO::Reset(void);
```

Reset sets the enumeration back to the first device.

Clone

```
HRESULT IEnumIMAGEIN_DEST_INFO::Clone(
      IEnumIMAGEIN_DEST_INFO **ppIEnum);
```

Clone creates another IEnumIMAGEIN_DEST_INFO enumeration object and returns an interface pointer to it.

Parameters:

ppIEnum	Address that receives a pointer to the new enumeration object.

IPropertyStorage Interface

The IPropertyStorage interface is used to query the device information and destination properties. IPropertyStorage is a standard OLE interface and is documented in the OLE Programmers Reference. All device information properties are read only. Any attempt to modify the device information properties will result in a failure with access denied. The methods of this interface are:

```
HRESULT  ReadMultiple(ULONG,  const PROPSPEC, PROPVARIANT);
HRESULT  WriteMultiple(ULONG,  const PROPSPEC, PROPVARIANT, PROPID);
HRESULT  DeleteMultiple(ULONG,  const PROPSPEC);
HRESULT  ReadPropertyNames(ULONG,  const PROPID, LPOLESTR);
HRESULT  WritePropertyNames(ULONG,  const PROPID, LPOLESTR);
HRESULT  DeletePropertyNames(ULONG,  const PROPID);
HRESULT  Commit(DWORD);
HRESULT  Revert(void);
HRESULT  Enum(IEnumSTATPROPSTG**);
HRESULT SetTimes(FILETIME const *, FILETIME const *, FILETIME const *);
HRESULT SetClass(REFCLSID);
HRESULT Stat(STATPROPSETSTG *);
```

Device Information Properties

All ImageIn devices support the following basic device information properties:

Device CLSID	- The CLSID of the ImageIn server for this device.
Unique Device ID	- A device ID which is unique per physical device.

Vendor Description	- The device manufactures name.
Device Description	- A description of the device.
Device Type	- A device type constant: scanner, camera...
Device Port Name	- Name of the port through which the device is connected.
Device Friendly Name	- A user readable device name.
Device Icon Resource	- Resource file name and ID.
Device Bitmap Resource	- Resource file name and ID.
Server Name	- The name of the server where the ImageIn server for this device is running.

ImageIn Scanner Device Object

ImageIn device objects support interfaces for querying and setting device properties, displaying device UI and transferring data. ImageIn devices are required to support a small number of standard interfaces that allow applications to deal with all devices in a common manner and transfer data from the devices in a manner that is native to the (COM) application. Device objects may also support more specialized interfaces to implement custom functions. Since the application has a direct connection to the device object, this architecture does not place any strict limits on the interfaces a device object can export. Practically speaking though, applications must know about an interface for it to be useful.

IScan Interface

ScanDlg

```
HRESULT ScanDlg(
        LONG                lFlags,
        LONG                lIntent);
```

ScanDlg presents the system or device UI needed to prepare a device for scanning. If the dialog successfully completes, the scan object will be ready to begin data transfer via the IDataObject or IimageTransfer interfaces. The default image capture properties may be specified using the optional IIntent parameter. Returns S_FALSE if the user cancels the selection dialog without making a device selection, S_OK if the user makes a device selection or an error value if the method fails.

Parameters:

IFlags	Operation flags
IIntent	Intent, default image capture properties

IPropertyStorage Interface

The standard IPropertyStorage interface is used to query and set all scan device properties. IPropertyStorage is a standard OLE interface and is documented in the OLE Programmers Reference. The methods of this interface are:

```
HRESULT  ReadMultiple(ULONG,  const PROPSPEC, PROPVARIANT);
HRESULT  WriteMultiple(ULONG,  const PROPSPEC, PROPVARIANT, PROPID);
HRESULT  DeleteMultiple(ULONG,  const PROPSPEC);
HRESULT  ReadPropertyNames(ULONG,  const PROPID, LPOLESTR);
HRESULT  WritePropertyNames(ULONG,  const PROPID, LPOLESTR);
HRESULT  DeletePropertyNames(ULONG,  const PROPID);
HRESULT  Commit(DWORD);
HRESULT  Revert(void);
HRESULT  Enum(IEnumSTATPROPSTG**);
HRESULT SetTimes(FILETIME const *, FILETIME const *, FILETIME const *);
HRESULT SetClass(REFCLSID);
HRESULT Stat(STATPROPSETSTG *);
```

All ImageIn scanner devices support the following basic device properties:

Horizontal Resolution	- The horizontal resolution in DPI.
Vertical Resolution	- The vertical resolution in DPI.
Horizontal Scan Start Position	- The horizontal start position in pixels.
Vertical Scan Start Position	- The vertical start position in pixels.
Horizontal Scan Extent	- The width of the scan in pixels.
Vertical Scan Extent	- The height of the scan in pixels.
Scan Data Type	- A data format specification constant.
Scan Color Depth	- The number of bits per pixel used to specify color.

ImageIn scan devices may support additional device properties depending on hardware configuration.

IDevPropStream Interface

The IDevPropStream interface is used to query/set all current device properties from/to a named, non-volatile, registry based storage. The methods of this interface are:

ReadDevPropStream

```
HRESULT ReadDevPropStream(BSTR bstrName);
```

ReadDevPropStream reads a device property stream from the specified value and initializes the device with these properties. The properties are per user and per device.

Parameters:

bstrName	Name of the property stream.

WriteDevPropStream

```
HRESULT WriteDevPropStream(BSTR bstrName);
```

WriteDevPropStream writes the current device property stream to the specified value. The properties are per user and per device.

Parameters:

bstrName	Name of the property stream.

ListDevPropStreams

```
HRESULT ListDevPropStreams(BSTR *pbstrName);
```

ListDevPropStreams lists the device property stream names present in non-volatile storage. The list is returned in an allocated BSTR which the application must free using SysFreeString.

Parameters:

pbstrName	Pointer to receive a list property streams.

DeleteDevPropStream

```
HRESULT DeleteDevPropStream(BSTR bstrName);
```

DeleteDevPropStream deletes a device property stream from non-volatile storage.
Parameters:

bstrName	Name of the property stream.

IDataObject Interface

The IDataObject interface is a standard OLE data transfer mechanism. This interface is used to provide an easy and natural way for applications to transfer data from a scan device. The full IDataObject interface description can be found in the OLE2 Programmer's Reference. The methods of this interface are:

```
HRESULT  GetData(LPFORMATETC, LPSTGMEDIUM);
HRESULT  GetDataHere(LPFORMATETC, LPSTGMEDIUM);
HRESULT  QueryGetData(LPFORMATETC);
HRESULT  GetCanonicalFormatEtc(LPFORMATETC,LPFORMATETC);
HRESULT  SetData(LPFORMATETC, STGMEDIUM FAR *, BOOL);
HRESULT  EnumFormatEtc(DWORD, LPENUMFORMATETC FAR *);
HRESULT  DAdvise(FORMATETC FAR *, DWORD,  LPADVISESINK, DWORD  FAR *);
HRESULT  DUnadvise(DWORD);
HRESULT  EnumDAdvise(LPENUMSTATDATA FAR *);
```

IImageTransfer Interface

The IImageTransfer is a high performance data transfer interface. This interface uses a shared memory window to transfer data from the device object to the application, eliminating unnecessary data copies during marshalling. For Simplicity, this interface uses the same format negotiation method as IDataObject. IImageTransfer uses two different mechanisms to transfer data.:

Banded Transfers

The device breaks a large image transfer up into smaller transfers, which are performed sequentially into an application-specified buffer.

Data Callback

The device does a single large transfer, while calling back to the application as the transfer progresses.

itAllocateTransferBuffer

```
HRESULT itAllocateTransferBuffer(
            LONG  lFlags,
            LONG  lSize,
            ULONG hSection,
            ULONG ProcessID);
```

Allocates a buffer to use for scanning data from the device. The buffer is mapped into the address space of both the client and server process.

Parameters:

lFlags	Operation flags.
lSize	Size in bytes of the requested buffer.
hSection	Mapped section handle
ProcessID	Caller process ID

itFreeTransferBuffer

```
HRESULT itFreeTransferBuffer(void);
```

Free the buffer created by itAllocateTransferBuffer. This buffer is also freed when the device object is destroyed.

itBeginTransfer

```
HRESULT itBeginTransfer(
            LPFORMATETC                 pFormatEtc,
            LONG                        lFlags,
            IDataCallback*              pIDataCallback);
```

Reset a device object context to begin an IImageTransfer.

Parameters:

pFormatEtc	Format specification.
lFlags	Operation flags.
pIDataCallback	Optional transfer progress notification entry point.

ItGetImage

```
HRESULT itGetImage(
            LONG                        lFlags,
            LONG*                       plSrcOffset,
            LONG*                       pcbWritten);
```

Perform an image transfer.

Parameters:

lFlags	Operation flags.
plSrcOffset	Source pointer for transfer.
pcbWritten	Actual number of bytes written during transfer.

itGetImageCB

```
HRESULT itGetImageCB(LONG lFlags);
```

Perform an image transfer using a user specified callback.

Parameters:

lFlags	Operation flags.

itEndTransfer

```
HRESULT itEndTransfer(LONG lFlags);
```

Cleanup a device object context after an image transfer.

Parameters:

lFlags	Operation flags.

itQueryGetData

```
HRESULT itQueryGetData(LPFORMATETC pfe);
```

Check to see if a device supports a given format in an image transfer.

Parameters:

pfe	Pointer to the FORMATETC structure defining the format, medium, and target device to use for the query.

itEnumFormatEtc

```
HRESULT itEnumFormatEtc(
            DWORD                           dw,
            LPENUMFORMATETC*                lpEnum);
```

Create a format enumerator.

Parameters:

dw	Reserved
lpEnum	Indirect pointer to the **IEnumFORMATETC** interface on the new enumerator object.

ImageIn Camera Device Object

The ImageIn Camera is a hierarchical object. The top-level camera object is used to get basic device information and also to get access to individual images or image folders. The CCameraItem object represents images and Image folders.

ICamera Interface

TakePicture

```
HRESULT  TakePicture();
```

The TakePicture method instructs a camera to take a new picture.

GetCameraRootItem

```
HRESULT  GetCameraRootItem(ICameraItem**  pICameraItem);
```

This method returns an interface to the root CCameraItem object. Using the IcameraItem interface it is possible to enumerate all images on the camera.

Parameters:

PICameraItem	Interface pointer to root camera item

GetCameraItemByHandle

```
HRESULT  GetCameraItemByHandle(
      ULONG              ulItemHandle,
      ICameraItem**            pICameraItem);
```

This method returns an ICameraItem interface to the specified CCameraItem object. A handle for a CCameraItem object can only be obtained through the ICameraItem interface.

Parameters:

ulItemHandle	Handle to camera item, previously returned by CameraItem interface
pICameraItem	Interface pointer to Camera Item specified by handle

CameraDlg

```
HRESULT  CameraDlg(
      LONG                     lFlags,
      LONG                     lIntent,
      ICameraItem**            pICameraItem);
```

Display the camera UI. UI will select an image on a device and prepare for image transfer. The UI returns a ICameraItem interface to a CCameraItem ready for data transfer. IDataObject or IImageTransfer interfaces are then used to transfer image data.

Parameters:

lFlags	Operational flags
lIntent	High level intent
pICameraItem	Interface pointer to Camera Item selected by Dlg

ICameraItem Interface

A CCameraItem object is created for every image and directory on the camera. This is done to support full and flexible hierarchical image storage on cameras. A ICameraItem pointer is returned for each CCameraItem object. ICameraItem methods are defined below.

GetItemType

```
HRESULT  GetItemType(ULONG *pItemType);
```

GetItemType returns the type of camera item the object represents. The two types of camera items are:

1. ItemTypeFolder - CCameraItem is a folder that may contain other CCameraItems
2. ItemTypeImage – CcameraItem is an image.

Parameters:

pItemType	Item Type returned

GetItemHandle

```
HRESULT  GetItemHandle(ULONG *pItemHandle);
```

Returns a handle to the CCameraItem. This handle can later be used by the ICamera interface to get an interface pointer back to this object.

Parameters:

PItemHandle Item handle returned. This handle may be used by the GetCameraItemByHandle method of the ICamera interface to obtain an IcameraItem pointer.

OpenImagePropertyStorage

```
HRESULT  OpenImagePropertyStorage(IPropertyStorage**
ppIPropertyStorage);
```

Returns an IPropertyStorage interface to the CCameraItem object.

Parameters:

ppIPropertyStorage	Returned IPropertyStorage interface pointer.

EnumChildItems

```
HRESULT  EnumChildItems(IEnumCameraItem** ppIEnumCameraItem);
```

Creates a CEnumCameraItem object and returns a IEnumCameraItem interface to it. This method only works if the camera item is a folder and the folder is not empty, as shown in the figure above.

Parameters:

ppIEnumCameraItem	Pointer to iEnumCameraItem interface.

IEnumCameraItem Interface

IEnumCameraItem is a standard OLE enumerator interface with the usual four methods Next, Skip, Clone and Reset. This enumerator will return an ICameraInterface for each camera item in the current camera folder (that was used in the call to EnumChildItems).

Next

```
HRESULT  Next(
        ULONG                   celt,
        ICameraItem**           ppICameraItem,
        ULONG*                  pceltFetched);
```

Return an ICameraItem interface for each CCameraItem contained by the current folder.

Parameters:

celt	number of camera items to get
ppICameraItem	List of iCameraItem interface pointers returned.
pceltFetched	Number of interface pointers returned

Skip

```
HRESULT  Skip(ULONG celt);
```

Skip celt CCameraItem objects.

Parameters:

celt	Number of CcameraItem objects to skip

Reset

```
HRESULT  Reset(void);
```

Begin at the first CCameraItem object.

Parameters: None

Clone

```
HRESULT  Clone(IEnumCameraItem **ppIEnumCameraItem );
```

Create a new CEnumCameraItem object and return a IEnumCameraItem interface.

Parameters:

ppIEnumCameraItem	New enumerator created to exactly match this one.

Camera Download Manager

The camera download manager monitors the connection status of camera devices. When a camera device is connected to the system, the Download Manager will attempt to immediately download all the images from the camera onto the system hard disk. The camera download manager will also maintain a data base of all images previously downloaded from a given camera and not download images that are already cached.

Caching images in this manner has several advantages.

- Camera image download can be very slow, this cache prevents any image from needing to be downloaded more than once.
- Applications do not need a special API to acquire images from a camera, simply wait for the download to complete and load the image from the file system.
- Images are safely and automatically copied to the file system where they can easily be edited, archived, printed or just saved.

In order for the camera download manager to work effectively, camera events must be detectable. For example, it should be easy to detect when a camera is connected or disconnected from the system. This should be detectable without loading the entire camera driver. Also any activity performed directly on the camera (through camera controls) must be reported to software so an accurate internal model of the state of the camera can be kept.

Sample Code

Get an image from an ImageIn device. Use the ImageIn device manager device selection UI and the device image acquisition UI.

```
/*****************************************************************************
*
*  GetImageFromImageIn
*
*  DESCRIPTION:
*   Use the ImageIn device manager to select a device and acquire an
*   image using the device UI.
*
*  PARAMETERS:
*   pszDIBfileName - Name of the DIB file which contain the image data.
*
*****************************************************************************/

HRESULT GetImageFromImageIn(LPOLESTR pszDIBfileName)
{
   HRESULT                  hr;
   IImageInDevMgr           *pIImageInDevMgr;

   // Get ImageIn device manager object.
   hr = CoCreateInstance(CLSID_ImageInDevMgr,
                         NULL,
                         CLSCTX_LOCAL_SERVER,
                         IID_IImageInDevMgr,
                         (void**)&pIImageInDevMgr);

   if (SUCCEEDED(hr)) {
      STGMEDIUM StgMedium;

      // Fill in the storage medium spec and get the image.
      StgMedium.tymed = TYMED_FILE;
      StgMedium.lpszFileName = pszDIBfileName;
      StgMedium.pUnkForRelease = NULL;
      hr = pIImageInDevMgr->GetImage(0,
                                     0,
                                     CF_DIB,
                                     &StgMedium);

      pIImageInDevMgr->Release();
   }
   return hr;
}
```

Enumerate all installed ImageIn devices and get the device ID of a specific device:

```
/*****************************************************************************
*
*  GetDeviceIDfromDescription
*
*  DESCRIPTION:
*   Get the unique device ID of an ImageIn device based on it's description.
*
*  PARAMETERS:
*   pszDescription   - Requested device description, from device info properties.
*   pszDeviceID      - Buffer for the returned device ID.
*   cbDeviceIDSize   - Size of the returned device ID buffer in bytes.
*
*****************************************************************************/

HRESULT GetDeviceIDfromDescription(
   LPOLESTR pszDescription,
   LPOLESTR pszDeviceID,
   UINT     cchDeviceID)
{
   HRESULT                hr;
   IImageInDevMgr         *pIImageInDevMgr;

   if (pszDeviceID && cchDeviceID) {
      *pszDeviceID = '\0';
   }
   else {
      return E_INVALIDARG;
   }

   // Get ImageIn device manager object.
   hr = CoCreateInstance(CLSID_ImageInDevMgr,
                         NULL,
                         CLSCTX_LOCAL_SERVER,
                         IID_IImageInDevMgr,
                         (void**)&pIImageInDevMgr);
   if (SUCCEEDED(hr)) {
      IEnumIMAGEIN_DEV_INFO *pIEnumIMAGEIN_DEV_INFO;
      PROPSPEC    PropSpec[1];
      PROPVARIANT PropVarDesc[1], PropVarID[1];
      UINT        cbSize;

      // Get an enumerator for the ImageIn device information.
      hr = pIImageInDevMgr->EnumImageInDevInfo(0, &pIEnumIMAGEIN_DEV_INFO);
      if (SUCCEEDED(hr)) {
         IPropertyStorage *pIPropStg;
         ULONG            ul;

         // Enumerate the ImageIn devices, getting a IImageInDevInfo
         // pointer for each. Use this interface to query the registry
         // based properties for each installed device.
         ul = 1;
         while ((hr = pIEnumIMAGEIN_DEV_INFO->Next(1,
                                                   &pIPropStg,
                                                   &ul)) == S_OK) {

            // Read the device description for the current device.
            PropSpec[0].ulKind = PRSPEC_PROPID;
            PropSpec[0].propid = DIP_DEV_DESC;
            hr = pIPropStg->ReadMultiple(1, PropSpec, PropVarDesc);
            if (SUCCEEDED(hr)) {
               // Test for a description match.
               if (!wcscmp(PropVarDesc[0].pwszVal, pszDescription)) {
                  // Read the device ID.
                  PropSpec[0].propid = DIP_DEV_ID;
                  hr = pIPropStg->ReadMultiple(1, PropSpec, PropVarID);
```

```
                if (SUCCEEDED(hr)) {
                    wcsncpy(pszDeviceID, PropVarID[0].pwszVal, cchDeviceID);
                    PropVariantClear(PropVarID);
                }
            }
            PropVariantClear(PropVarDesc);
        }
        pIPropStg->Release();
        if (*pszDeviceID) {
            break;
        }
    }
    pIEnumIMAGEIN_DEV_INFO->Release();
  }
  pIImageInDevMgr->Release();
  if (!*pszDeviceID) {
    hr = E_FAIL;
  }
 }
 return hr;
}
```

United States Patent & Trademark Office

Office of Initial Patent Examination -- Scanning Division

Application deficiencies found during scanning:

☐ Page(s)_____of_____ were not present for scanning. (Document title)

☐ Page(s)_____of_____were not present for scanning. (Document title)

☐ ***Scanned copy is best available.*** drawing figs 1 to 7 are dark